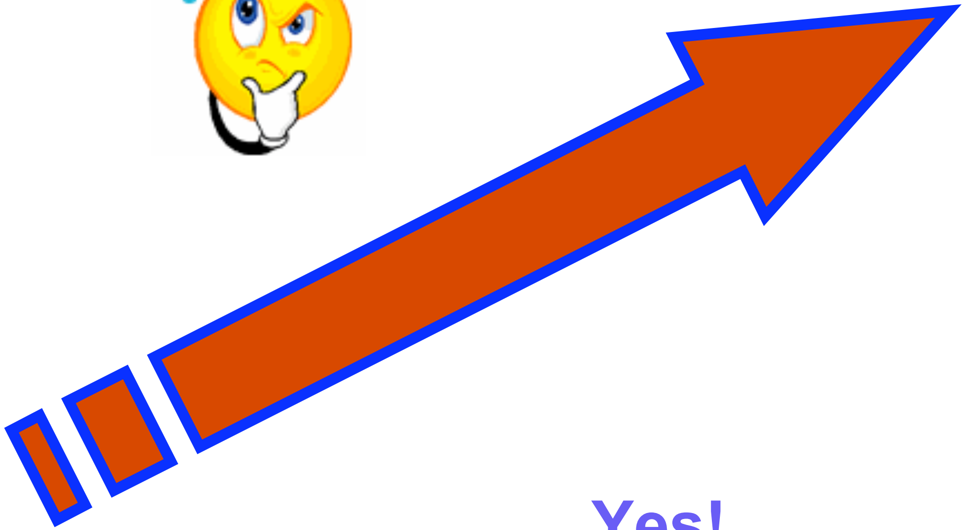
ALCF & Blue Gene Consortium
WORKSHOP SERIES

BG/L → BG/P Transition: An Applications Perspective

Rajiv Bendale, Kirk E. Jordan, Jerrold Heyman, Carlos P Sosa, Robert E. Walkup
IBM, USA
bendale@us.ibm.com
kjordan@us.ibm.com
jheyman@us.ibm.com
cpsosa@us.ibm.com
walkup@us.ibm.com

Is There a Need to Transition My Application?

Blue Gene/P
PPC 450 @ 850MHz
Scalable to 3+ PF



Yes!

Blue Gene/L
PPC 440 @ 700MHz
Scalable to 596+ TF

Where Are the Differences?

Environment

- **Hardware**
- **Software**

Application

- **Compiling and linking**
- **Running**

Selected Hardware Features

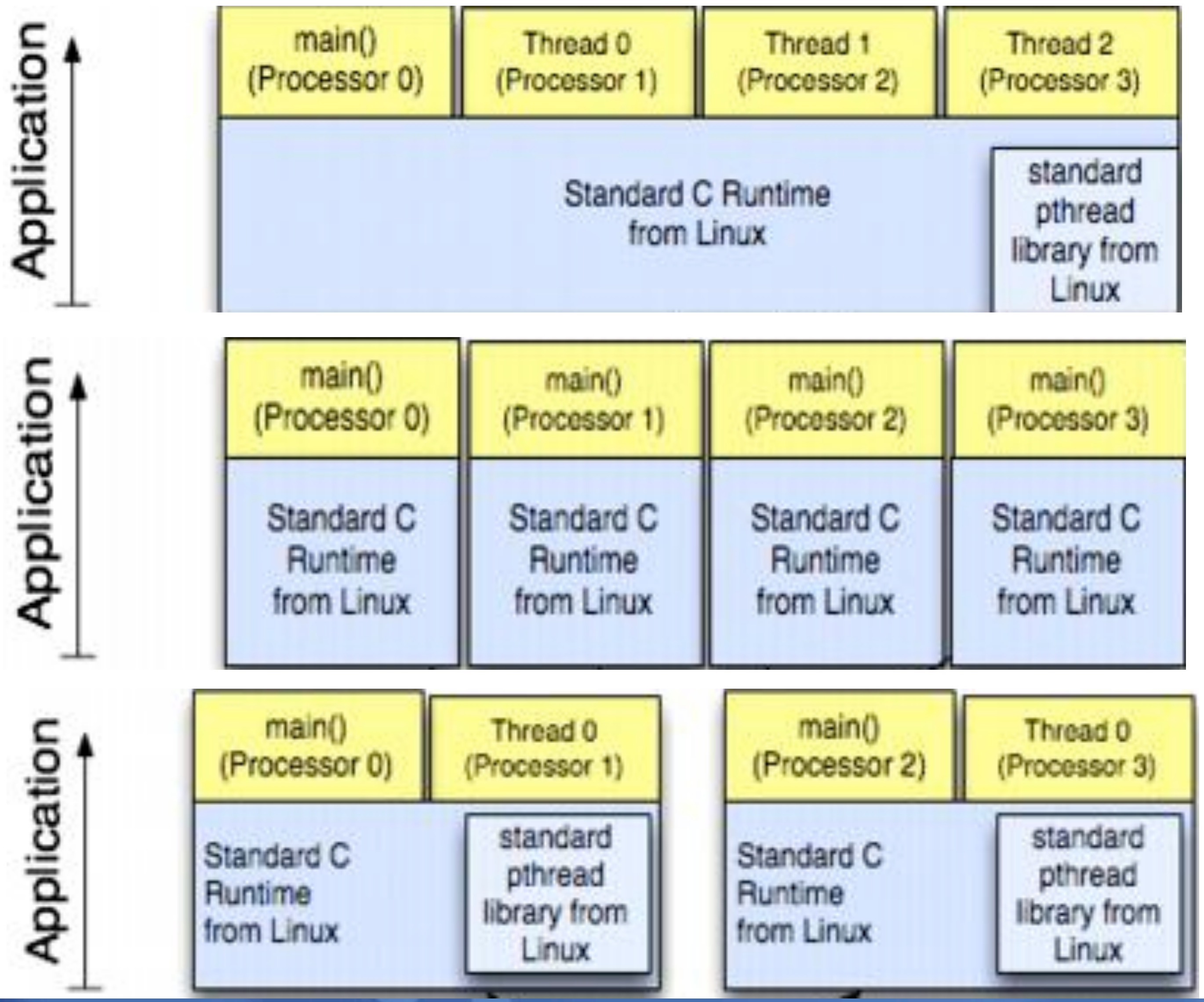
Feature	BG/L	BG/P
Cores per node	2	4
Core clock speed	700 MHz	850 MHz
Physical memory per node	512 MB – 1 GB	2 GB
Peak performance, per node	5.6 GFlop/Sec	13.6 GFlop/Sec

Running Your Application

Execution Process Modes

BG/L	BG/P
Co-processor (CO) mode	Symmetrical Multiprocessing (SMP) Node Mode
Virtual node (VN) mode	Virtual Node Mode (VN)
	Dual Node Mode (DUAL)

SMP, VN, and DUAL

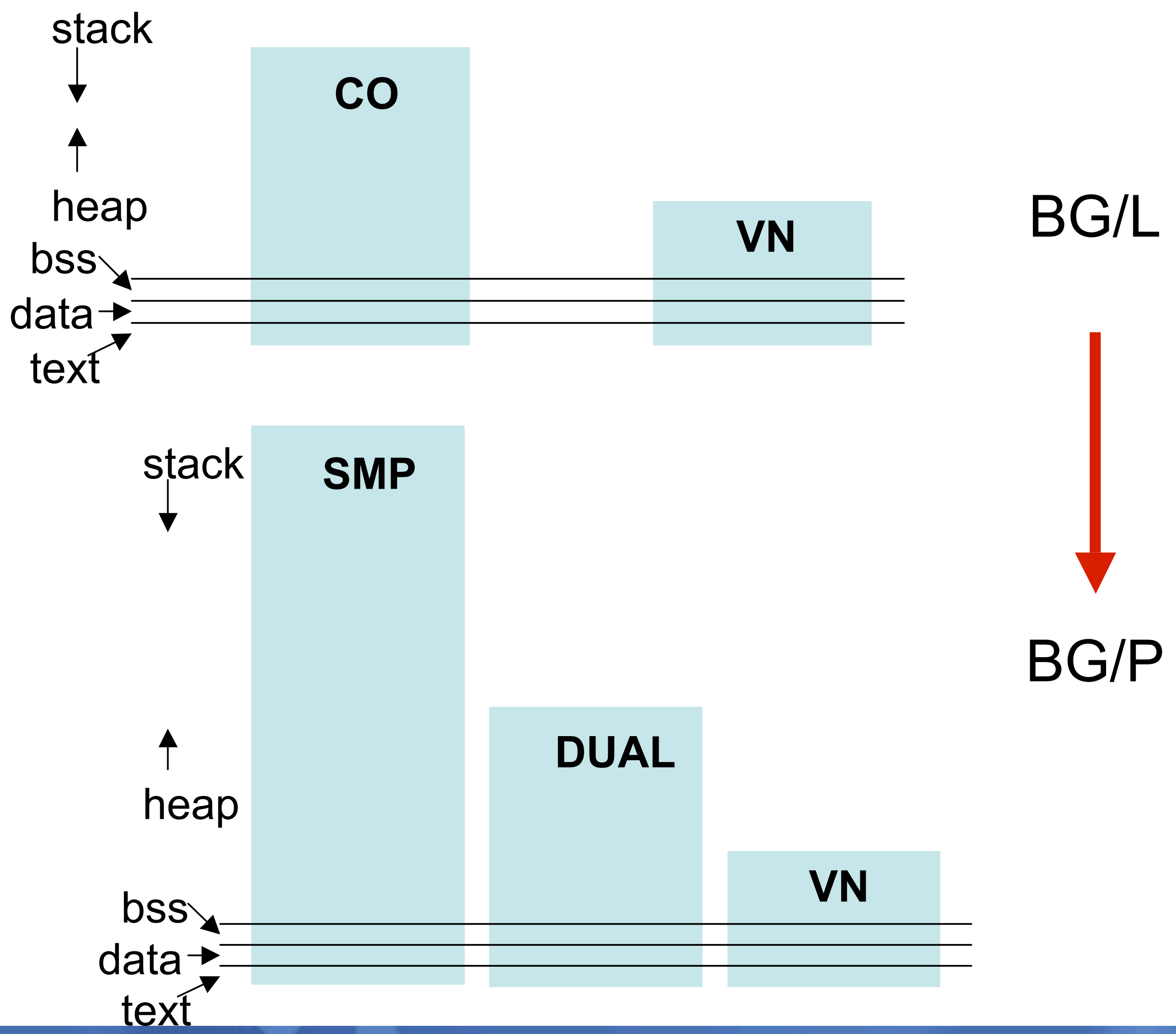

Application
main()
(Processor 0)
Thread 0
(Processor 1)
Thread 1
(Processor 2)
Thread 2
(Processor 3)
Standard C Runtime from Linux
standard pthread library from Linux
Application
main()
(Processor 0)
main()
(Processor 1)
main()
(Processor 2)
main()
(Processor 3)
Standard C Runtime from Linux
Standard C Runtime from Linux
Standard C Runtime from Linux
Standard C Runtime from Linux
Application
main()
(Processor 0)
Thread 0
(Processor 1)
Standard C Runtime from Linux
standard pthread library from Linux
main()
(Processor 2)
Thread 0
(Processor 3)
Standard C Runtime from Linux
standard pthread library from Linux

SMP

VN

DUAL

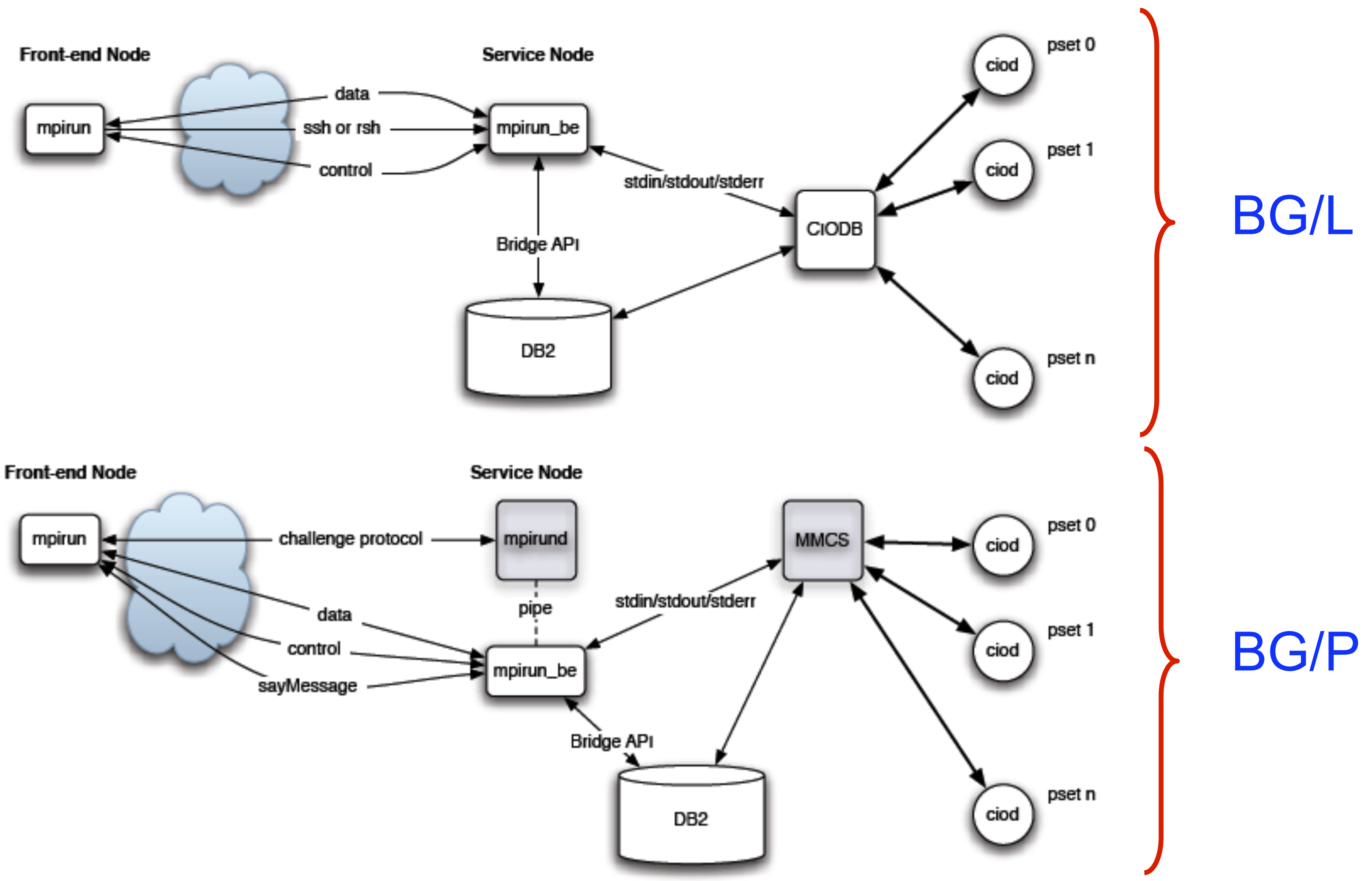
Memory Addressing: BG/L → BG/P



Choosing Execution Mode

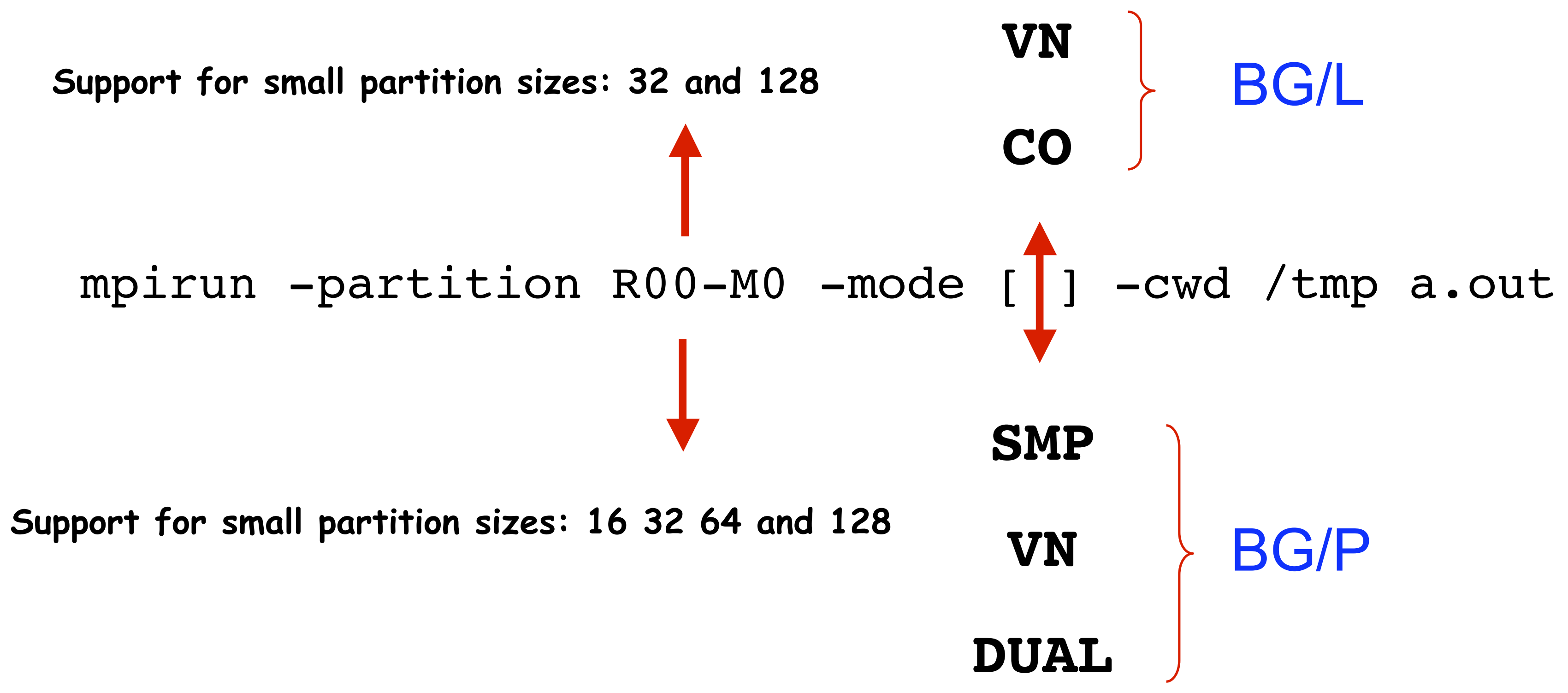
- **It is application dependent**
  - Applications based on a hybrid parallel paradigm (MPI+OpenMP) may benefit from the SMP node mode
  - Single threaded applications may consider VN node mode
  - Applications that are CPU bound and do not have large memory requirements may benefit from VN

mpirun Developers Differences



Invoking mpirun

```
mpirun [options]
```

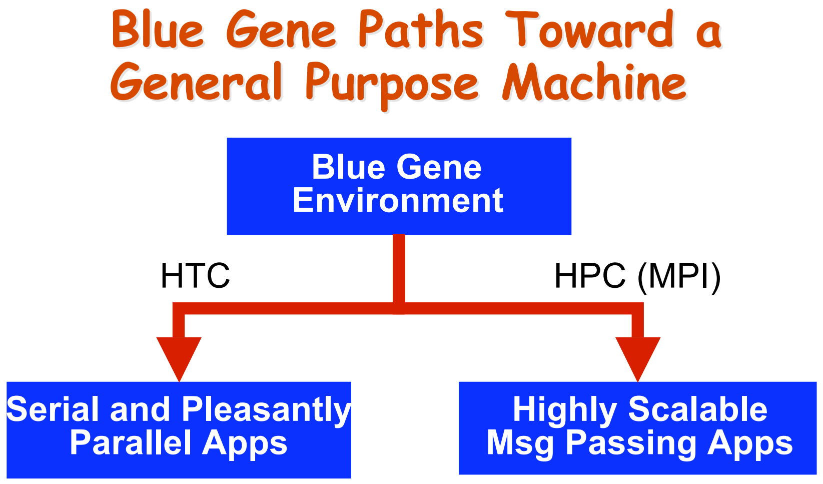


```
mpirun -partition R00-M0 -mode [ ] -cwd /tmp a.out
```

Support for small partition sizes: 32 and 128
VN
CO
BG/L
Support for small partition sizes: 16 32 64 and 128
SMP
VN
DUAL
BG/P

High Throughput Computing Mode: BG/L $\rightarrow$ BG/P

High-Throughput Computing Mode (HTC)


Blue Gene Paths Toward a General Purpose Machine
Blue Gene Environment
HTC
HPC (MPI)
Serial and Pleasantly Parallel Apps
Highly Scalable Msg Passing Apps

Benefits of High-Throughput Computing (HTC) mode on Blue Gene:

- Blue Gene looks like a traditional "cluster" from an application’s point of view
- Enables a new class of workloads that use many single-node jobs
- Blue Gene supports hybrid application environment, traditional HPC (MPI) and now HTC apps

BG/P HTC Features

- Provides a job ***submit*** command that is simple, lightweight, and extremely fast
- Job state is integrated into MMCS, so users know which nodes have jobs, and which are idle
- Provides stdin/stdout/stderr on a per-job basis
- Enables individual jobs to be signaled or killed
- Maintains a user ID on per-job basis (allows multiple users per partition)
- Navigator shows HTC jobs (active or in history) with job exit status & runtime stats
- Leverages support on I/O node to use an I/O daemon (CIOD) per compute node
- Designed for easy integration with job schedulers

How Does It Work?

- “submit” client:
  - Acts as a shadow or proxy for the real job running on the compute node
  - Similar to mpirun, but lightweight
  - Submit jobs to location or pool
    - Pool id concept: scheduler alias for a collection of partitions available to run a job on (pool id defaults to partition name if not set)
    - location: the resource where the job will execute in the form of a processor or wildcard location

Example #1 (submit to location): `submit -location “R00-M0-N00-J05-C00” -exe hello_world`

Example #2 (submit to pool): `submit -pool BIOLOGY –exe hello_world`

- Job scheduler:
  - Submit jobs using Condor (“condor_submit”)
  - Submit jobs using SIMPLE (“qsub”)

What Applications Fit the HTC Mode?

- **Wide range of applications can run in HTC mode**
  - Many applications that run on Blue Gene today are “embarrassingly (pleasantly) parallel” or “independently parallel”
  - They don’t exploit the torus for MPI communication and just want a large number of small tasks, with a coordinator of results

Grid.org

HTC Application Identification

- **Solution Statement:**
  - A high-throughput computing (HTC) application is one in which the same basic calculation must be performed over many independent input data elements and the results collected. Because each calculation is independent, it is extremely easy to spread calculations out over multiple cluster nodes. For this reason, high-throughput applications are sometimes called “embarrassingly parallel.” HTC applications occur much more frequently than one might think, showing up in areas such as parameters studies, search applications, data analytics, and what-if calculations.
- **Identifying a HTC application:**
  - There are a number of identifiers you can use to determine if your specific computing problem fits into the category of a high-throughput application:
  - Do you need to run many instances of the same application with different arguments or parameters?
  - Do you need to run the same application many times with different input files?
  - Do you have an application that can select subsets of the input data and whose results can be combined by a simple merge process such as concatenating, placing them into a single database, or adding them together?

If the answer to any of these questions is “yes,” then it is quite likely that you have a HTC application.

Compiling Your Application

Compilers Information on BG/P

- **XL C/C++ Advanced Edition V9.0 for Blue Gene**
  - http://www.ibm.com/software/awdtools/xlcpp/library/
- **XL Fortran Advanced Edition V11.1 for Blue Gene**
  - http://www-306.ibm.com/software/awdtools/fortran/xlfortran/library/
- **You can also find these documents in the following directories:**
  - /opt/ibmcmp/vacpp/bg/9.0/doc (C and C++)
  - /opt/ibmcmp/xlf/bg/11.1/doc (Fortran)
- **The compilers can be found in the following directories:**
  - /opt/ibmcmp/vac/bg/9.0/bin
  - /opt/ibmcmp/vacpp/bg/9.0/bin
  - /opt/ibmcmp/xlf/bg/11.1/bin

Compiler Wrappers: BG/L → BG/P

- **Blue Gene/P release:**
  - blrts_ is replaced by bg.
  - xlf 11.1, vacpp 9.0, and vac 9.0 on the Blue Gene/L system support both blrts_ and bg
  - -qarch=450d/450 is for the Blue Gene/P system
  - -qarch=440d/440 is for the Blue Gene/L system

Compiler Scripts

Language	Script Name	
	BG/L	BG/P
C	blrts_xlc	bgc89, bgc99, bgcc, bgxlc bgc89_r, bgc99_r bgcc_r, bgxlc_r
C++	blrts_xlc++	gxlc++, bgxlc++_r, bgxlC, bgxlC_r
Fortran	blrts_xlf, blrts_xlf90, blrts_xlf95	bgf2003, bgf95, bgxlf2003, bgxlf90_r, bgxlf_r, bgf77, bgfort77, bgxlf2003_r, bgxlf95, bgf90, bgxlf, bgxlf90, bgxlf95_r

Compiler BG/P Release: New

- **Full support for the OpenMP 2.5 standard**
- **Use of the same infrastructure as the OpenMP that is supported on AIX and Linux**
- **Interoperability with MPI**
  - MPI at outer level, across the Compute Nodes
  - OpenMP at the inner level, within a Compute Node
- **Autoparallelization based on the same parallel execution framework**
  - Enablement of autoparallelization as one of the loop optimizations
  - Thread-safe version for each compiler: bgxlf_r, bgxlc_r, bgxlC_r, bgcc_r
- ***The thread-safe compiler version should be used with any threaded, OpenMP, or SMP application***
- **Usage of -qsmp and -qthreaded OpenMP and SMP applications**
  - -qsmp by itself automatically parallelizes loops
  - -qsmp=omp automatically parallelizes based on OpenMP directives in the code.
  - -qsmp=omp:noauto -qthreaded should be used when parallelizing codes manually. It prevents the compiler from trying to automatically parallelize loops

http://publib.boulder.ibm.com/infocenter/comphelp/v8v101/index.jsp

Shared Memory Parallelism (SMP)

- **The SMP features in the XL compilers were disabled in XLC v8 and XLF 10.1**
  - mainly because of the lack of coherent L2 on BG/L
- **BG/P supports 4 cores per node and has a coherent L2 cache**
- **The compiler can take advantage of the SMP capabilities of the BG/P in two ways**
  - Parallelization via user-inserted SMP or OMP directives
  - Automatic loop parallelization

SMP Selected Options Information

- **By default, the runtime will use all available processors**
  - Maximum of 4 on **BG / P**
  - Do not set the PARTHDS or OMP_NUM_THREADS variables unless you wish to use fewer than the number of available processors. [Currently SMP Mode]

SMP Example: source

```
#include <omp.h>

long long timebase(void);

int main(argc, argv)
int argc;
char *argv[];
{
   int num_threads;
   long n, i;
   double area, pi, x;
   long long time0, time1;
   double cycles, sec_per_cycle,
  factor;
       n    = 1000000000;
       area = 0.0;
       time0 = timebase();
```

```
#pragma omp parallel for
  private(x) reduction(+: area)
       for (i = 0; i < n; i++) {
          x = (i+0.5)/n;
          area += 4.0 / (1.0 +
  x*x);
       }
       pi = area / n;
       printf ("Estimate of pi:
  %7.5f\n", pi);
       time1 = timebase();
       cycles = time1 - time0;
       factor = 1.0/850000000.0;
       sec_per_cycle = cycles *
  factor;
       printf("Total time %lf
  \n",sec_per_cycle, "Seconds
  \n");
}
```

SMP Example: Makefile

```
BGP_FLOOR    = /bgsys/drivers/ppcfloor
BGP_IDIRS    = -I$(BGP_FLOOR)/arch/include -I$(BGP_FLOOR)/comm/include
BGP_LIBS     = -L$(BGP_FLOOR)/comm/lib -L$(BGP_FLOOR)/runtime/SPI -lmpich.cnk -ldcmfcoll.cnk -
ldcmf.cnk -lrt -lSPI.cna -lpthread

XL           = /opt/ibmcmp/vac/bg/9.0/bin/bgxlc_r

EXE          =  pi_reduction_bgp
OBJ          =  pi_reduction.o
SRC          =  pi_reduction.c
FLAGS        =  -O3 -qsmp=omp:noauto -qthreaded -qarch=450  -qtune=450 -
I$(BGP_FLOOR)/comm/include
FLD          =  -O3 -qarch=450 -qtune=450

$(EXE): $(OBJ)
#       ${XL} $(FLAGS) $(BGP_LDIRS) -o $(EXE) $(OBJ) $(BGP_LIBS)
        ${XL} $(FLAGS) -o $(EXE) $(OBJ) timebase.o $(BGP_LIBS)
$(OBJ): $(SRC)
        ${XL} $(FLAGS) $(BGP_IDIRS) -c $(SRC)

clean:
        rm pi_reduction.o pi_reduction_bgp
```

SMP Example: Run script

```
#!/bin/csh

set MPIRUN="mpirun"
set MPIOPT="-np 1"
set MODE="-mode SMP"
set PARTITION="-partition N06_32_1"
set WDIR="-cwd /bgusr/cpsosa/red/pi/c"
set EXE="-exe /bgusr/cpsosa/red/pi/c/pi_reduction_bgp"
#
$MPIRUN $PARTITION $MPIOPT $MODE $WDIR $EXE -env "OMP_NUM_THREADS=4"
#
echo "That's all folks!!"
```

BG/P

BG/P

Parallel Speedup

Threads	Elapsed Time in Sec.
1	
POWER4 1 GHz	**20.12**
POWER5 1.9 GHz	**5.22**
POWER6 4.7 GHz	**4.78**
BG/P	**12.80**
2	
POWER4 1 GHz	**10.08**
POWER5 1.9 GHz	**2.70**
POWER6 4.7 GHz	**2.41**
BG/P	**6.42**
4	
POWER4 1 GHz	**5.09**
POWER5 1.9 GHz	**1.40**
POWER6 4.7 GHz	**1.28**
BG/P	**3.24**

OpenMP vs. Automatic Parallelization

- **Compiler automatically detects parallel regions and inserts OMP directives.**
  - No user intervention required
  - Some user-insert assertions (e.g disjoint) may help the compiler identify parallelizable loops
- **User-inserted OMP directives and auto-parallelization can co-exist**
  - The compiler will only auto-parallelize loops that are not marked by OMP directives
- ***Automatic parallelization along with -qreport can be helpful for identifying parallel loop opportunities for an OpenMP programmer***

SMP / OMP Stack Overflow Checking *(-qsmp=stackcheck)*

- **A new SMP suboption “-qsmp=stackcheck” is introduced along with the environment variable XLSMPOPTS="stackcheck[=n]" for stack overflow checking in SMP/OMP codes.**
  - where "n" is the stack overflow warning limit (in bytes). The default value of "n" is 4096 bytes.
- **Applies only to the slave threads, not master.**
- **The local stack for slave threads is allocated by the SMP runtime, hence it is possible to warn the user when stack overflow is imminent.**
- **Stack overflow checking can be very useful for figuring out if stack overflow was the reason an abnormal termination of the application. The compiler will warn the users when the remaining stack size is less than “n” bytes.**

Automatic Parallelization Control Threshold *(-qsmp=threshold=n)*

- **A new SMP feature in V9 / 11.1**
- **The suboption -qsmp=threshold=n is introduced to control the amount of automatic loop parallelization that occurs.**
  - Where 'n' is a positive integer. The default value is 100, which means parallelize only the profitable auto-parallel loops. 0 implies parallelize all auto-parallelizable loops whether or not it is profitable. Values greater than 100 would result in more loops getting serialized. A large enough value of 'n' could end up serializing all the loops.
- **This sub-option affects only auto parallel loops. User parallel (OMP/SMP) loops are not affected.**
  - -qsmp=threshold=n will enable SMP with all the defaults options, which include auto.
  - -qsmp=omp:threshold=n will enable SMP, but the explicit omp will disable auto and will make threshold have no effect

Better SIMD reports

- **The -qreport option produces a list of high level transformation performed by the compiler**
  - Everything from unrolling, loop interchange, SIMD transformations, etc.
  - Also contains transformed "pseudo source"
- **All loops considered for SIMDization are reported**
  - Successful candidates are reported
  - If SIMDization was not possible, the reasons that prevented it are also provided
- **Can be used to quickly identify opportunities for speedup**

Examples of SIMD Messages

- **Loop was not SIMD vectorized because it contains operation which is not suitable for SIMD vectorization.**
- **Loop was not SIMD vectorized because it contains function calls.**
- **Loop was not SIMD vectorized because it is not profitable to vectorize.**
- **Loop was not SIMD vectorized because it contains control flow.**
- **Loop was not SIMD vectorized because it contains unsupported vector data types**
- **Loop was not SIMD vectorized because the floating point operation is not vectorizable under -qstrict.**
- **Loop was not SIMD vectorized because it contains volatile reference**

Debugging at High Optimization Level *(-qoptdebug)*

- **High level optimizations may transform source such that it has little resemblance to the original**
- **The -qoptdebug option is introduced to allow source-level debug of compiler generated pseudo-C or pseudo-Fortran**
  - The compiler generates post-optimized pseudo source files
  - The debug information in the executable is changed to refer to these new psuedo sources instead of the original
  - This option only has an effect at optimization levels -O3 and above
- **The pseudo code will be dumped to <*source file*>*.o.optdbg*. When the users load their program in a debugger, the debugger will show the compiler generated pseudo source code, instead of the original C/C++/Fortran source.**

Example (-qoptdebug)

```
#include <math.h>
void foo (float * restrict a, float * restrict b)
{
    for (int i=0; i < 100; i++)
      a[i] = cos(b[i])
}

void main(void)
{
   float *a, *b;
   foo (a, b);
}
```

```
Breakpoint 1, foo(a=0xffadeb60, b=0x1002b84) at  z.o.optdbg:3
3          8 |   $.NumElements0 = (long) 100u;
(gdb) list
1          3 |  void foo (char * a, char * b)
2          4 |  {
3          8 |   $.NumElements0 = (long) 100u;
4                 __vscos(((char *)a + (4)*(0)),((char *)b + (4)*(0)),&$.NumElements0)
5         10 |   return;
6               } /* function */
7
8
9         12 |  void main()
10        13 |  {
(gdb)
```

- Program z.c compiled as -O3 -g -qoptdebug, producing new file z.o.optdbg
- Binary loaded into debugger as   gdb ./a.out
- Caveats:

  Does not eliminate the well known limitations of debugging optimized code

  Optimizations performed by low level optimizer may continue to inhibit debugging

MASS Enhancements: BG/L and BG/P

- **Mathematical Acceleration SubSystem is a library of highly tuned, machine specific, mathematical functions available for download from IBM**
  - Contains both scalar and vector versions of many (mostly trig.) functions
  - Trades off very limited accuracy for greater speed
  - The compiler tries to automatically vectorize scalar math functions and generate calls to the MASS vector routines in libxlopt
  - Failing that, it tries to inline the scalar MASS routines (new for this release)
  - Failing that, it generates calls to the scalar routines instead of those in libm
- **More info: http://www-306.ibm.com/software/awdtools/mass/bgl/**

MASS Example

```
subroutine mass_example (a,b)
   real     a(100), b(100)
   integer          i

   do i = 1, 100
      a(i) = sin(b(i))
   enddo;
end subroutine mass_example
```

$\longrightarrow$

-O3 –qhot -qreport

```
SUBROUTINE mass_example (a, b)
  @NumElements0 = int(100)
    CALL __vssin (a, b, &@NumElements0)
  RETURN
END SUBROUTINE mass_example
```

Aliasing prevents vectorization:

```
void c_example(float *a, float *b)
{
    for (int i=0; i < 100; i++)
    {
       a[i] = sin(b[i]);
       b[i] = (float) i;
    }
}
```

```
void c_example(float *a, float *b)
{
    @CIV0 = 0;
    do {
       a[@CIV0] = __xl_sin(b[@CIV0]);
       b[@CIV0] = (float) @CIV0;
       @CIV0 = @CIV0 + 1;
    } while ((unsigned) @CIV0 < 100u);
    return;
}
```

Unsupported Options

- **BG/L**
  - -qsmp
  - -q64
  - -qaltivec
  - -qpic
  - -qmkshrobj
- **BG/P**
  - -q64
  - -qaltivec

GNU Compiler Collection

- **The standard GNU Compiler Collection 4.1.1 for C, C++, and Fortran is supported on the Blue Gene/P system:**
  - gcc 4.1.2
  - binutils 2.17
  - glibc 2.4

Linking

- **BG/L**
  - Dynamic linking is not supported; only static linking
- **BG/P**
  - Static and dynamic linking

Dynamic Linking

- **Instead of embedding all of the dependent library routines into the executable, only a small stub is used instead. This reduces the size of the file on disk.**
- **XLC V9 / XLF 11.1 supports dynamic linking on BG/P**
  - Not available on **BG/L**
  - Default remains static linking on both **BG/L** and **BG/P**
- **Just add the following option to the compile/link command:**

  C/C++: -qnostaticlink

  Fortran: -Wl,-dy
- **Make dynamic libraries by compiling with -qpic and linking the objects with appropriate GNU linker flags, same as on LoP**

XL Static Linking Example: BG/L and BG/P

```
#!/bin/csh
#
# Compile with the XL compiler
/opt/ibmcmp/vac/bg/9.0/bin/bgxlc -c pi.c
/opt/ibmcmp/vac/bg/9.0/bin/bgxlc -c main.c
#
# Create library
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-ar
rcs libpi.a pi.o
#
# Create executable
/opt/ibmcmp/vac/bg/9.0/bin/bgxlc -o pi main.o -L. -lpi
```

GNU Static Linking Example: BG/L and BG/P

```
#!/bin/csh
#
# Compile with the GNU compiler
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -c pi.c
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -c main.c
#
# Create library
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-ar rcs libpi.a pi.o
#
# Create executable
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -o pi main.o -L. -lpi
```

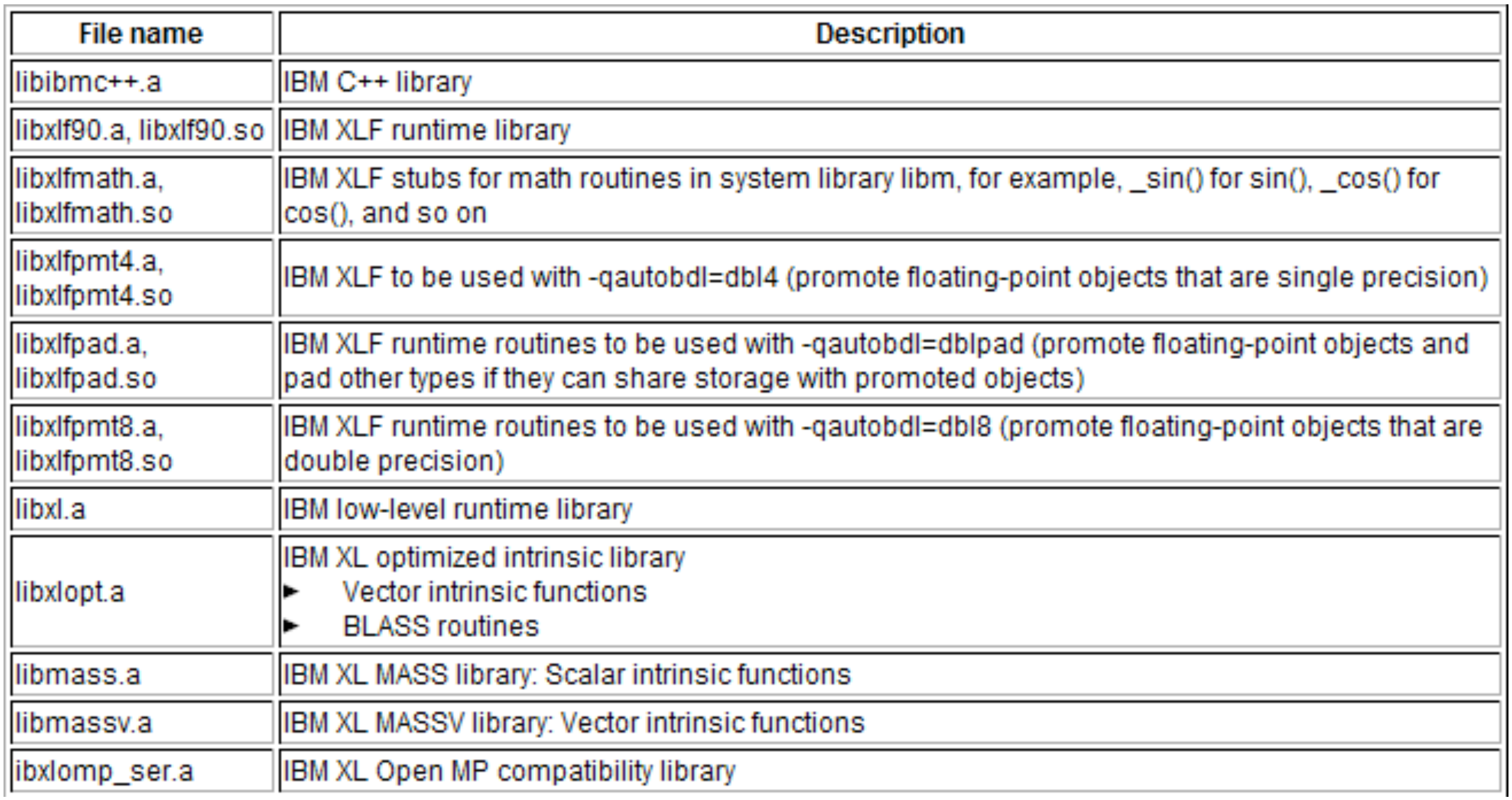
XL Dynamic Linking Example: BG/P

```
#!/bin/csh
#
# Use XL to create shared library
/opt/ibmcmp/vac/bg/9.0/bin/bgxlc -qpic -c libpi.c
/opt/ibmcmp/vac/bg/9.0/bin/bgxlc -qpic -c main.c
#
# Create shared library
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -shared \
                    -Wl,-soname,libpi.so.0 -o libpi.so.0.0 libpi.o -lc
#
# Set up the soname
ln -sf libpi.so.0.0 libpi.so.0
#
# Create a linker name
ln -sf libpi.so.0 libpi.so
#
# Create executable
/opt/ibmcmp/vac/bg/9.0/bin/bgxlc -o pi main.o -L. -lpi -qnostaticlink
```

GNU Dynamic Linking Example: BG/P

```
#!/bin/csh
#
# Compile with the GNU compiler
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -fPIC -c libpi.c
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -fPIC -c main.c
#
# Create shared library
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -shared \
                    -Wl,-soname,libpi.so.0 -o libpi.so.0.0 libpi.o -lc
#
# Set up the soname
ln -sf libpi.so.0.0 libpi.so.0
#
# Create a linker name
ln -sf libpi.so.0 libpi.so
#
# Create executable
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc-bgp-linux-gcc -o pi main.o -L. -lpi -dynamic
```

Static and Dynamic Libraries



File name	Description
libibmc++.a	IBM C++ library
libxlf90.a, libxlf90.so	IBM XLF runtime library
libxlfmath.a, libxlfmath.so	IBM XLF stubs for math routines in system library libm, for example, _sin() for sin(), _cos() for cos(), and so on
libxlfpmt4.a, libxlfpmt4.so	IBM XLF to be used with -qautobdl=dbl4 (promote floating-point objects that are single precision)
libxlfpad.a, libxlfpad.so	IBM XLF runtime routines to be used with -qautobdl=dblpad (promote floating-point objects and pad other types if they can share storage with promoted objects)
libxlfpmt8.a, libxlfpmt8.so	IBM XLF runtime routines to be used with -qautobdl=dbl8 (promote floating-point objects that are double precision)
libxl.a	IBM low-level runtime library
libxlopt.a	IBM XL optimized intrinsic library ► Vector intrinsic functions ► BLASS routines
libmass.a	IBM XL MASS library: Scalar intrinsic functions
libmassv.a	IBM XL MASSV library: Vector intrinsic functions
ibxlomp_ser.a	IBM XL Open MP compatibility library

Compilation and Run Environments

- **Front-End Node for Compilation:**
  - BG/L:

    SUSE Linux Enterprise Server 9 Service Pack 3 (SLES9 SP3) for IBM POWER

    GCC 3.3.3
  - BG/P:

    SUSE Linux Enterprise Server 10 Service Pack 1 (SLES10 SP1) for IBM POWER

    GCC 4.1.2
- **Application Execution**
  - BG/L: GNU Toolchain built for Blue Gene/L based on gcc 3.4.3 and glibc 2.3.6
  - BG/P: GNU Toolchain built for Blue Gene/P based on gcc 4.1.2 and glibc 2.4

BlueGene specific predefined macros

- Aim to predefine same macros as the corresponding gcc toolchain for compatibility.
- To distinguish BGL vs BGP:

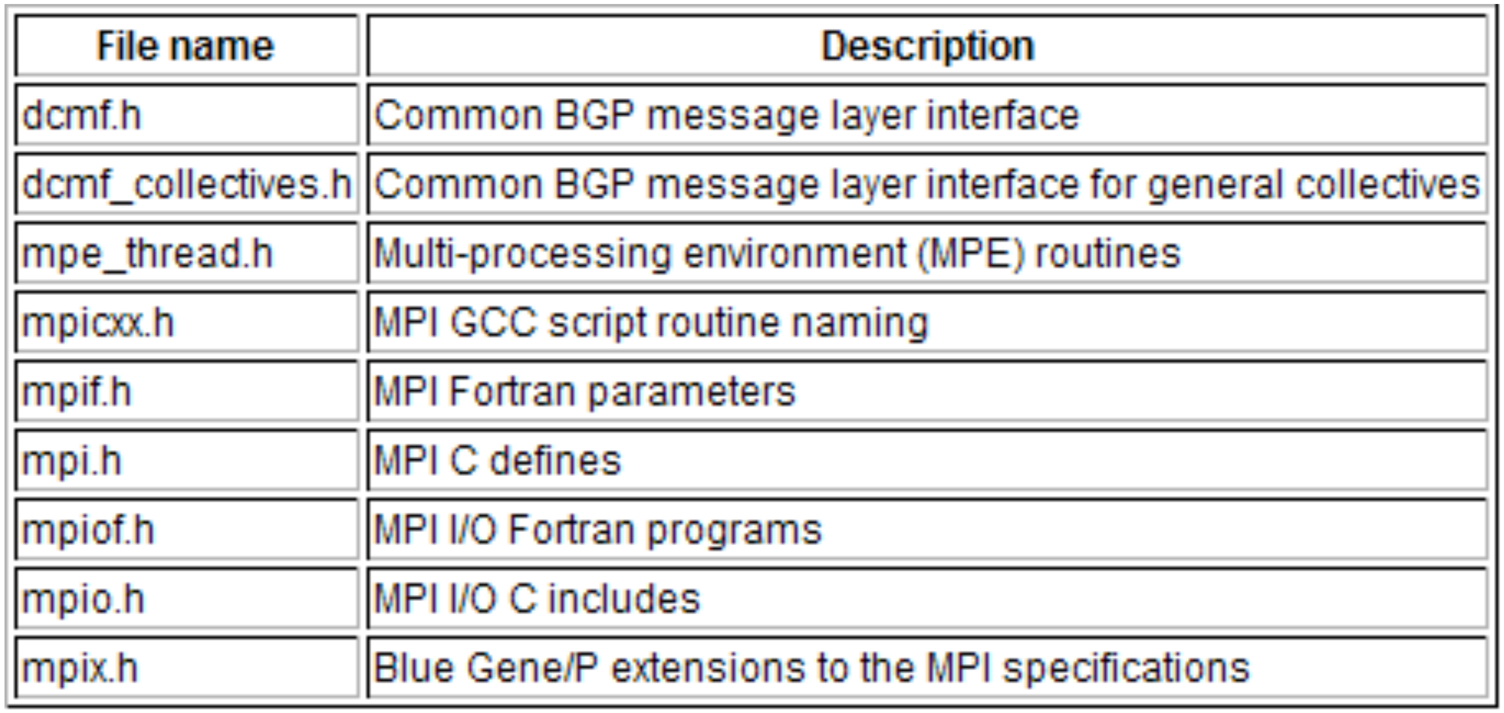
  __bg__ predefined on BGL/BGP

  __bgp__ predefined on BGP

  __blrts__ predefined on BGL
- Complete list documented in "Using the XL compilers for BlueGene"

BG/P Headers files in /bgsys/drivers/ppcfloor/comm/include

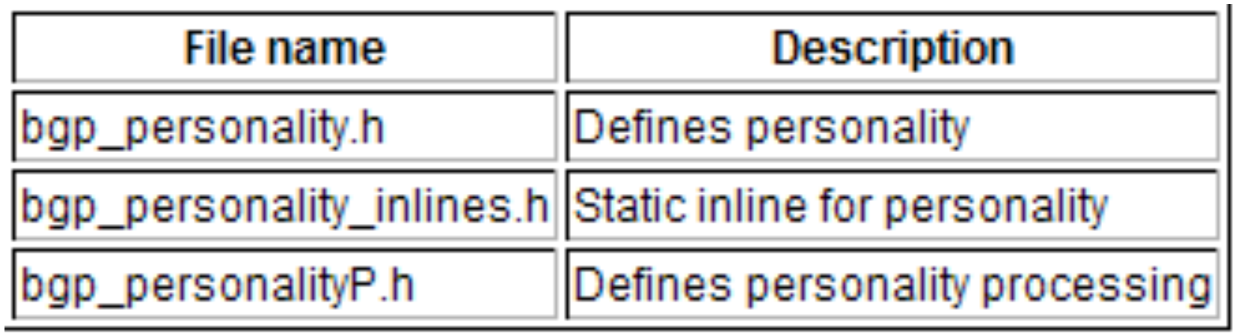
Location: /bgsys/drivers/ppcfloor/comm/include



File name	Description
dcmf.h	Common BGP message layer interface
dcmf_collectives.h	Common BGP message layer interface for general collectives
mpe_thread.h	Multi-processing environment (MPE) routines
mpicxx.h	MPI GCC script routine naming
mpif.h	MPI Fortran parameters
mpi.h	MPI C defines
mpiof.h	MPI I/O Fortran programs
mpio.h	MPI I/O C includes
mpix.h	Blue Gene/P extensions to the MPI specifications

Headers Files in /bgsys/drivers/ppcfloor/arch/include/common

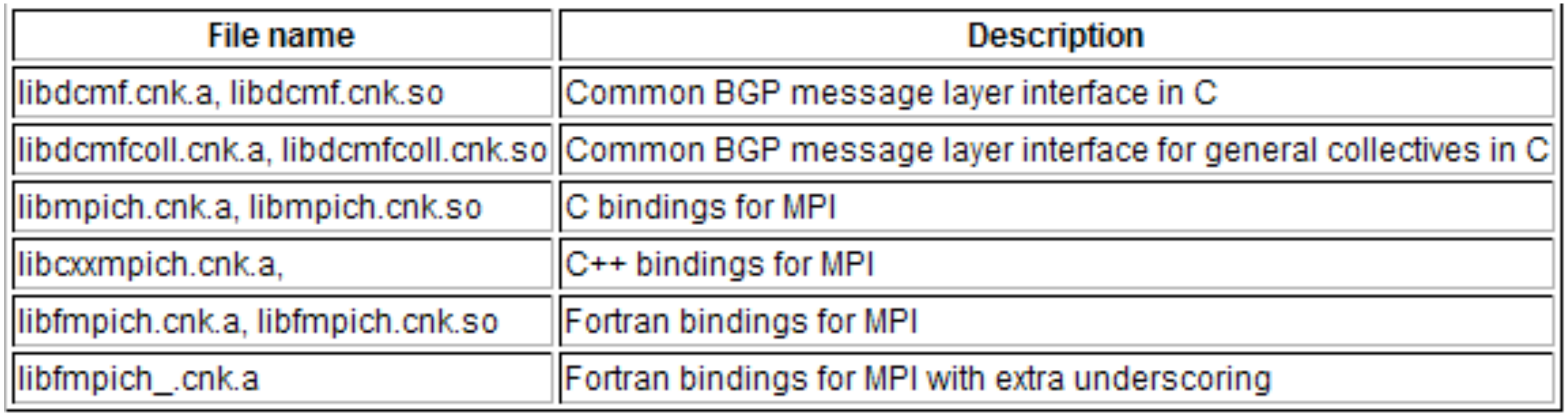
Location: /bgsys/drivers/ppcfloor/arch/include/common



File name	Description
bgp_personality.h	Defines personality
bgp_personality_inlines.h	Static inline for personality
bgp_personalityP.h	Defines personality processing

32-bit static and dynamic libraries in /bgsys/drivers/ppcfloor/comm/lib/

Location: /bgsys/drivers/ppcfloor/comm/lib



File name	Description
libdcmf.cnk.a, libdcmf.cnk.so	Common BGP message layer interface in C
libdcmfcoll.cnk.a, libdcmfcoll.cnk.so	Common BGP message layer interface for general collectives in C
libmpich.cnk.a, libmpich.cnk.so	C bindings for MPI
libcxxmpich.cnk.a,	C++ bindings for MPI
libfmpich.cnk.a, libfmpich.cnk.so	Fortran bindings for MPI
libfmpich_.cnk.a	Fortran bindings for MPI with extra underscoring

System Calls Supported by the Compute Node Kernel

- **System calls supported on BG/L**
  - IBM System Blue Gene Solution: Application Development: SG24-7179, Chapter 3
  - http://www.redbooks.ibm.com/redbooks/SG247179/
- **System calls supported on BG/P**
  - IBM System Blue Gene Solution: Blue Gene/P Application Development: SG24-7287
  - http://www.redbooks.ibm.com/redbooks/SG247287/

References: http://www.redbooks.ibm.com/redbooks

- *IBM System Blue Gene Solution: Blue Gene/P Application Development* , SG24-7287
- *Blue Gene Safety Considerations*, REDP-4257
- *Blue Gene/L: Hardware Overview and Planning*, SG24-6796
- *Blue Gene/L: Performance Analysis Tools*, SG24-7278
- *Evolution of the IBM System Blue Gene Solution*, REDP-4247 *GPFS*
- *Multicluster with the IBM System Blue Gene Solution and eHPS Clusters*, REDP-4168
- *IBM System Blue Gene Solution: Application Development*, SG24-7179
- *IBM System Blue Gene Solution: Configuring and Maintaining Your Environment*, SG24-7352
- *IBM System Blue Gene Solution: Hardware Installation and Serviceability*, SG24-6743
- *IBM System Blue Gene Solution Problem Determination Guide*, SG24-7211
- *IBM System Blue Gene Solution: System Administration*, SG24-7178
- *Unfolding the IBM eServer Blue Gene Solution*, SG24-6686
- Blue Gene/P System and Optimization Tips*
- Recommendations for Porting Open Source Software (OSS) to Blue Gene/P*

Additional Information

Debugging Applications on BG/P

- **Four pieces of code are involved when debugging applications on the Blue Gene/P system:**
  - The Compute Node Kernel, which provides the low-level primitives that are necessary to debug an application
  - The control and I/O daemon (CIOD) running on the I/O Nodes, which provides control and communications to Compute Nodes
  - A "debug server" running on the I/O Nodes, which is vendor-supplied code that interfaces with the CIOD
  - *A debug client running on a Front End Node, which is where the user does their work interactively*

GNU Project Debugger

- **More info:**
  - http://www.gnu.org/software/gdb/gdb.html
    http://www.gnu.org/software/gdb/documentation/
- **Support has been added to the Blue Gene/P system for which the GDB can work with applications that run on Compute Nodes**
- **IBM provides a simple debug server called *gdbserver***
- **Each running instance of GDB is associated with one, and only one, Compute Node**
- **If you must debug an MPI application that runs on multiple Compute Nodes, and you must, for example, view variables that are associated with more than one instance of the application, you run multiple instances of GDB.**

Core Processor Debugger

- **Core Processor is a basic tool that can help you debug your application**
- **This tool is discussed in detail in *IBM System Blue Gene Solution: Blue Gene/P System Administration*, SG24-7417**

addr2line Utility

- **The addr2line utility is a standard Linux program**
- **You can find additional information about this utility in any Linux manual as well as at the following Web site:**
  - http://www.linuxcommand.org/man_pages/addr2line1.html
- **The addr2line utility translates an address into file names and line numbers**